论文分享

MARL同时“学习”和“建议”

林越 2022.9.6

Simultaneously Learning and Advising in Multiagent Reinforcement Learning

Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa
Escola Politécnica of the University of São Paulo, Brazil
{f.leno,ruben.glatt,anna.reali}@usp.br

- AAMAS 2017

摘要1：问题和以前的方法

- 问题：要和环境互动很多次才能有足够的数据去学好策略
    - 在MA里这个问题更严重
- 可能的解决方法1：The teacher-student approach (Transfer Learning)
    - 做法：在student agent学习的时候，有一个有经验的teacher student(fixed policy)来给student agent建议，来引导student agent的探索
    - 局限：要求teacher student是个expert
- 可能的解决方法2：分享成功的episode
    - 局限：communication要求高，在一些领域communication是受限的

摘要2：此论文的新方法

- 提出了个multi agent advising framework
  - 每个agent在学习的时候都可以给别人建议（自己没训完也可以建议别人）
  - agent在任何时候，如果不知道怎么做，可以问别人建议
  - 会收到对答案（此时该采取什么动作）更有自信的agent的答案
  - 不一定是MARL，每个agent可以是别的学习算法
  - 做的是纯合作的，奖励每个人都一样；推荐的建议动作一定在询问者的动作空间
  - 提的框架也可以适用self-interested agent，要多加机制来识别advisor，此文还没做

The Teacher-Student Framework

专家teacher给student建议，加速student的学习

- framework (M. E. Taylor 2009)
  - teacher能看到student的学习过程（student的obs），任何时候都可给建议
    - teacher的策略固定
  - 建议有个预算次数，建议次数超过这个次数后就不能再给建议
    - teacher要决定什么时候给建议，才能最好地加快进程
  - teacher确定值得给建议后，student一定要听（直接采取被建议的动作）

The Teacher-Student Framework

teacher怎么知道什么时候建议student比较好

- 一个teacher一个student(L. Torrey and M. E. Taylor 2013)
  - teacher用的TD，teacher有一个importance metric $I(s)$
  - $$I(s) = \max_a Q_{teacher}(s,a) - \min_a Q_{teacher}(s,a)$$
    - **Importance Advising**：如果当前的$I(s)$超过了预先定义的一个阈值，那么就建议
    - **Mistake Correcting Advice**：如果此时student要采取一个在teacher看来是错误的动作，那么teacher才建议（文中没说错误指的是什么，我猜是$I(s)$的第二个Q里的a改成student采取的a，然后也是对比阈值）

The Teacher-Student Framework

teacher怎么知道什么时候建议student比较好

- 多个teacher (Y. Zhan 2016)
  - 也要求**teacher的策略是固定的**，也能在student学习时给建议
  - student听到多个teacher的建议，但可以拒绝听建议（不做被建议的动作）

- 用在MARL里的话，teacher也会是学习的，不满足固定策略的设定

The Teacher-Student Framework

这篇文章认为teacher没学好的时候就给别人建议，也是可以加速学习的

- 这篇文章说teacher不一定要先学好
    - 这篇文章提出了边学边建议别人（文章标题就是这个）
- 别人建议我，在以下情况中是有用的：
    - 我这state没探索过，别人探索过了
    - 我刚来这环境，别人已经在这环境里学了一会了
    - 我的学习算法没别人的高效（每个人的学习算法不一定一样）

Advising Strategy

识别哪些agent有好策略，然后让他去建议别人

- 边学边建议，每个人都可以是teacher也可以是student
- 每个step都建立一个ad hoc advisor-advisee关系，ad hoc=临时
  - 建立这个关系是根据每个agent在自己当前state下的**自信程度**
  - 每个人都有 $\langle P_{ask}, P_{give}, b_{ask}, b_{give}, G, \Gamma \rangle$ （这里的state好像其实都是obs）
    - $P_{ask} : S \to [0, 1]$ 一个state对应一个问建议的概率（随着训练会越来越小）
    - $P_{give} : S \to [0, 1]$ 这就是自信程度，也是在这个state下被问到后给建议的概率（随着训练会越来越大）

Advising Strategy

识别哪些agent有好策略，然后让他去建议别人

- 边学边建议，每个人都可以是teacher也可以是student
- 每个step都建立一个ad hoc advisor-advisee关系，ad hoc=临时
  - 建立这个关系是根据每个agent在自己当前state下的**自信程度**
  - 每个人都有 $\langle P_{ask}, P_{give}, b_{ask}, b_{give}, G, \Gamma \rangle$
    - 两个b都是预算buget，可以问别人的次数，可以给建议的次数
    - G是当前state下我能接触到的别的agent的集合（和我有这个临时关系的人）
    - Γ是函数，当有多个人给我建议，我采取哪个建议动作来执行

Advising Strategy

识别哪些agent有好策略，然后让他去建议别人

- 边学边建议，每个人都可以是teacher也可以是student
- 每个step都建立一个ad hoc advisor-advisee关系，ad hoc=临时
  - 这个关系是一个这样的有序对$\langle i, j, s_i, \zeta, \pi_j \rangle$，给完建议之后关系结束
  - 问建议的人i，advisee；给建议的人j，adviser
  - advisee的决策依据，state或者observations_i
  - ζ是个函数，adviser试图理解advisee看到了什么，输入s_i，输出作为π_j的输入
  - π_j指的是advisor给建议的策略，建议是一个advisee的动作：$a_i = \pi_j(\zeta(s_i))$

ζ是什么

- 让adviser知道advisee看到了什么（应该是想表达这两人算法不同也可以）
- 如果是fully observable的环境
    - 原话是exchanging the state features of the advisee with the advisor without the need for additional communication
    - 应该是直接给
- 如果是partially observable的环境
    - advisee要告诉adviser看到了啥，ζ是个communicate过程

Advising Strategy

episode生成的流程：advisee 询问的人

Algorithm 1 Action selection for a potential advisee i

Require: advising probability function P_{ask}, budget b_{ask}, action picker function Γ, confidence function Υ.

for all training steps **do**
  Observe current state s_i.
  if $b_{ask} > 0$ **then**
    $p_{s_i} \leftarrow P_{ask}(s_i, \Upsilon)$
    With probability p_{s_i} **do**
      Define reachable agents $G(s_i)$.
      $\Pi \leftarrow \emptyset$
      for $\forall z \in G(s_i)$ **do**
        $\Pi \leftarrow \Pi \cup z.advice(s_i)$
      if $\Pi \neq \emptyset$ **then**
        $b_{ask} \leftarrow b_{ask} - 1$
        $a \leftarrow \Gamma(\Pi)$
        Execute a.
  if no action was executed in this step **then**
    Perform usual exploration strategy.

- 如果询问的budget没用完
- 查看自己在当前观测下的**自信程度**，来决定选择问别人的概率
- 在这个概率下，如果选择问别人，则：
  - 看哪些人我能问得到（看通信）
  - 让这些人采取建议
  - 如果有人采取建议，那么我budget-1，然后在众多建议中选一条来执行（majority vote[34]）
- 没人给建议，或者是我选择执行自己策略：就执行自己策略

Advising Strategy

episode生成的流程：advisee 询问的人

Algorithm 1 Action selection for a potential advisee i

Require: advising probability function P_{ask}, budget b_{ask}, action picker function Γ, confidence function Υ.

for all training steps **do**
  Observe current state s_i.
  if $b_{ask} > 0$ **then**
    $p_{s_i} \leftarrow P_{ask}(s_i, \Upsilon)$
    With probability p_{s_i} **do**
      Define reachable agents $G(s_i)$.
      $\Pi \leftarrow \emptyset$
      for $\forall z \in G(s_i)$ **do**
        $\Pi \leftarrow \Pi \cup z.advice(s_i)$
      if $\Pi \neq \emptyset$ **then**
        $b_{ask} \leftarrow b_{ask} - 1$
        $a \leftarrow \Gamma(\Pi)$
        Execute a.
  if no action was executed in this step **then**
    Perform usual exploration strategy.

- 如果不知道哪些人是reachable的，可以广播，发给所有人，如果别人能收到那就是reachable的

Advising Strategy

episode生成的流程：advisee 询问的人

Algorithm 1 Action selection for a potential advisee i

Require: advising probability function P_{ask}, budget b_{ask}, action picker function Γ, confidence function Υ.

for all training steps **do**
  Observe current state s_i.
  if $b_{ask} > 0$ **then**
    $p_{s_i} \leftarrow P_{ask}(s_i, \Upsilon)$
    With probability p_{s_i} **do**
      Define reachable agents $G(s_i)$.
      $\Pi \leftarrow \emptyset$
      for $\forall z \in G(s_i)$ **do**
        $\Pi \leftarrow \Pi \cup z.advice(s_i)$
      if $\Pi \neq \emptyset$ **then**
        $b_{ask} \leftarrow b_{ask} - 1$
        $a \leftarrow \Gamma(\Pi)$
        Execute a.
  if no action was executed in this step **then**
    Perform usual exploration strategy.

- $P_{ask}(s, \Upsilon) = (1 + \upsilon_a)^{-\Upsilon(s)}$
  - 这个像海鸥一样的符号叫upsilon，有普西隆，表示自信程度；大于等于0
  - 自信程度越高，问别人的可能性就越低：P_{ask}递减
  - υ_a是个超参，越大则问别人的概率越低
- $\Upsilon_{visit}(s) = \sqrt{n_{visits}(s)}$
  - 这是个例子，任意学习算法都可以用这个函数；假设其策略会随着学习过程而变得更好

Advising Strategy

episode生成的流程：adviser 给建议的人

Algorithm 2 Response to advice requirement.

Require: advising probability function P_{give}, budget b_{give}, advisor policy π, advisee state s_i, state translation function ζ, confidence function Ψ.

if $b_{give} > 0$ **then**
  $p_{s_j} \leftarrow P_{give}(s_i, \Psi)$
  With probability p_{s_j} **do**
    $b_{give} \leftarrow b_{give} - 1$
    return $\pi(\zeta(s_i))$
return $\emptyset$

- 如果给建议的budget没用完
- 查看自己在advisee观测下的**自信程度**，来决定给advisee建议的概率
- 在这个概率下，如果选择该建议，则：
  - 我给建议的budget-1
  - 然后执行我给建议的策略

Advising Strategy

episode生成的流程：adviser 给建议的人

Algorithm 2 Response to advice requirement.

Require: advising probability function P_{give}, budget b_{give}, advisor policy π, advisee state s_i, state translation function ζ, confidence function Ψ.

if $b_{give} > 0$ **then**
  $p_{s_j} \leftarrow P_{give}(s_i, \Psi)$
  With probability p_{s_j} **do**
    $b_{give} \leftarrow b_{give} - 1$
    return $\pi(\zeta(s_i))$
return $\emptyset$

- $P_{give}(s, \Psi) = 1 - (1 + \upsilon_g)^{-\Psi(s)}$
  - \phi也是自信程度；大于等于0
  - 自信程度越高，给别人建议的可能性就越高：P_{give}递增
  - υ_g是个超参，越大则问别人的概率越低
- $\Psi_{visit}(s) = log_2 n_{visits}$任意都可以用
- $\Psi_{TD}(s) = \Upsilon_{visit}(s) \, |max_a Q(s,a) - min_a Q(s,a)|$
  - importance advising，TD agent可用

Advising Strategy

两种advising

- visit-based advising
  - $\Upsilon_{visit}(s) = \sqrt{n_{visits}(s)}$
  - $\Psi_{visit}(s) = log_2 n_{visits}$
- temporal difference advising
  - $\Upsilon_{visit}(s) = \sqrt{n_{visits}(s)}$
  - $\Psi_{TD}(s) = \Upsilon_{visit}(s)\ |max_a Q(s,a) - min_a Q(s,a)|$

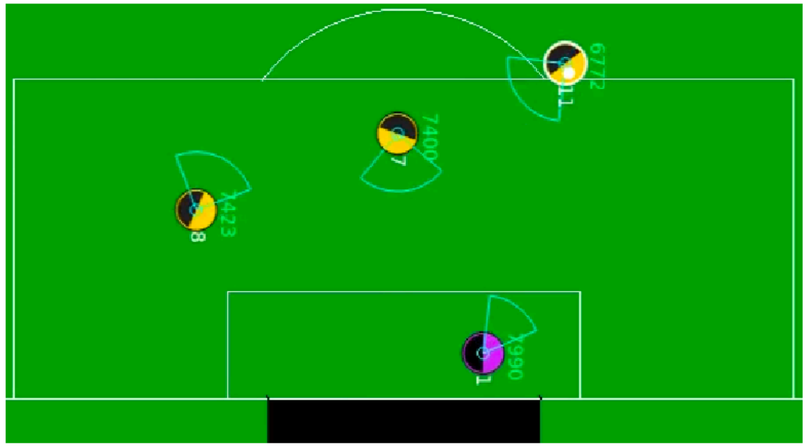
实验

测试的算法

- 上一页的两个算法
    - **visit-based advising + SARSA(λ)**
    - **temporal difference advising + SARSA(λ)**
- **Adapted Importance-Based Teacher-Student Advising + SARSA(λ)**
    - 改进前的，每个agent都是student，都有一个teacher对应，teacher知道student的任意时刻的state；关系是长久的（具体实现不知道）
- **Episode Sharing + SARSA(λ)**
    - 每有一个成功的episode，就可分享；分享一个transitions要扣一个单位的budget
- **independent SARSA(λ)**

实验

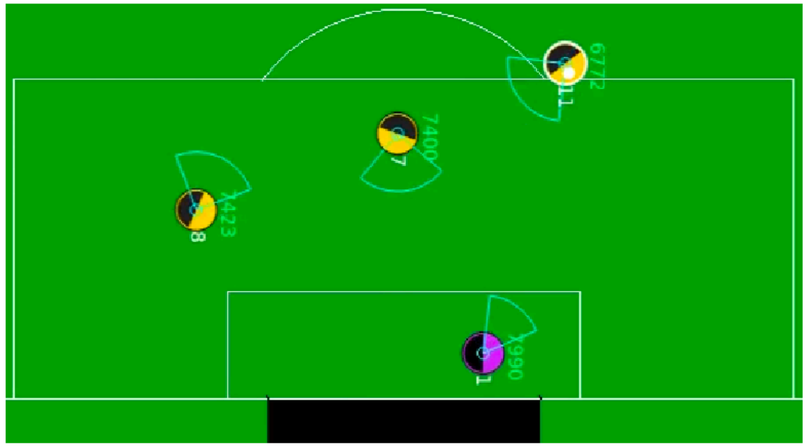
环境



- Half Field Offense (HFO) 半场踢足球进攻的游戏
  - 完全合作，自己写算法控制三个人进攻
  - HFO是个简化版的RoboCup任务，守门员是2012年RoboCup 2D冠军队伍的算法Helios policy控制的
  - 进球了每人都+1奖励，球离场或者被守门员拿到就每人-1

实验

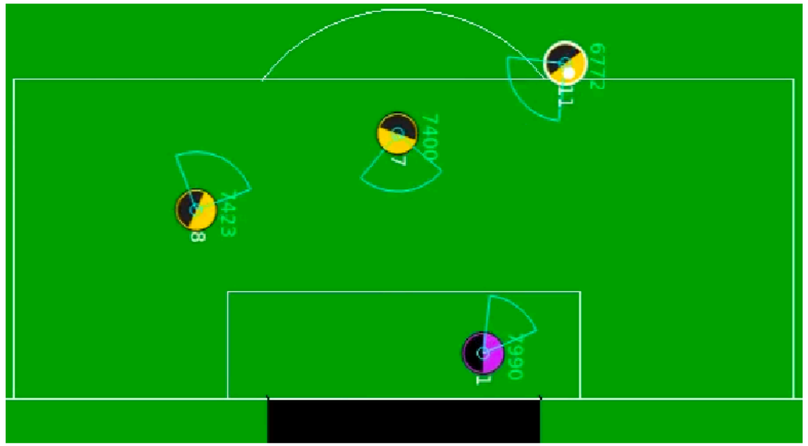
环境



- Half Field Offense (HFO) 半场踢足球进攻的游戏
- 三个agents的动作空间
    - 没球时的动作：Move（这个动作是自动的，朝着Helios策略的最佳位置移动过去）
    - 有球时的动作：射门、传球到最远的友方、传球到最近的友方（一共就3个人）、运球向球门跑
- 三个agents的状态空间
    - 能不能踢（我有没有球）、球门中心点、我的位置对球门中心的角度、如果没人拦着时我能得分的最大角度（以及其他两个队友的这个数据）

实验

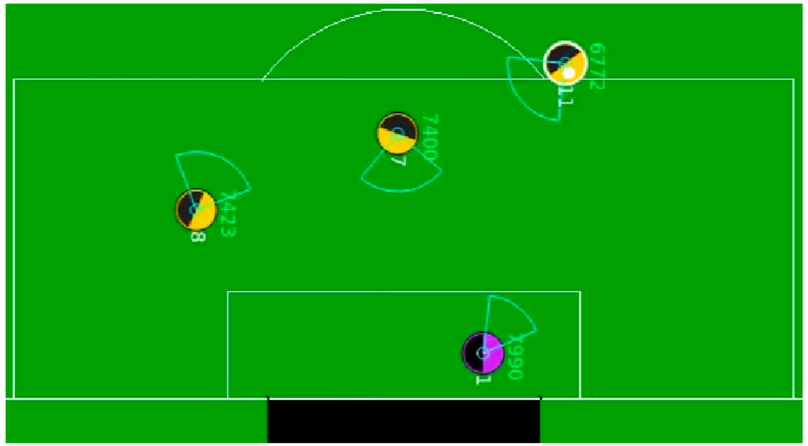
细节



- 先每个人练5000个episode
- 然后每训练20个episode就评价一次性能
    - 评价时不更新、不随机探索、不能问别人
    - 评价要用当前的policy跑100个episode
- 实验有两个场景
    - 场景1：3个agent都从头开始学（还研究了Mistake Correcting Advice）
    - 场景2：让其中1个agent先学3000个steps

实验

评价指标

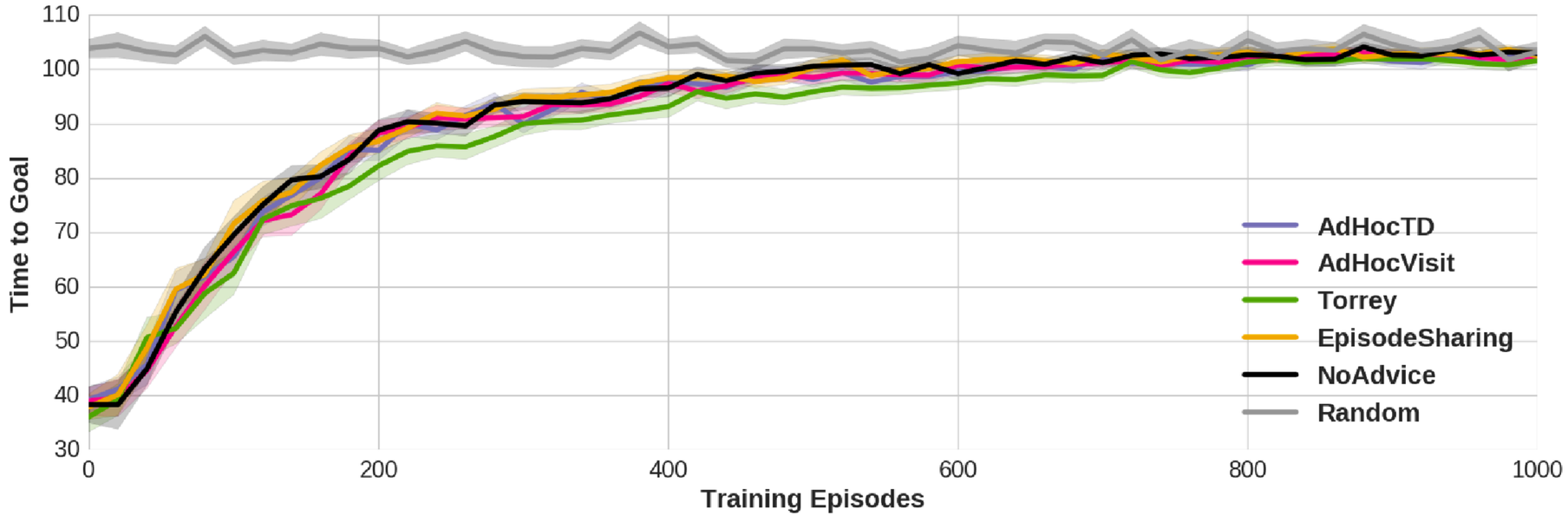


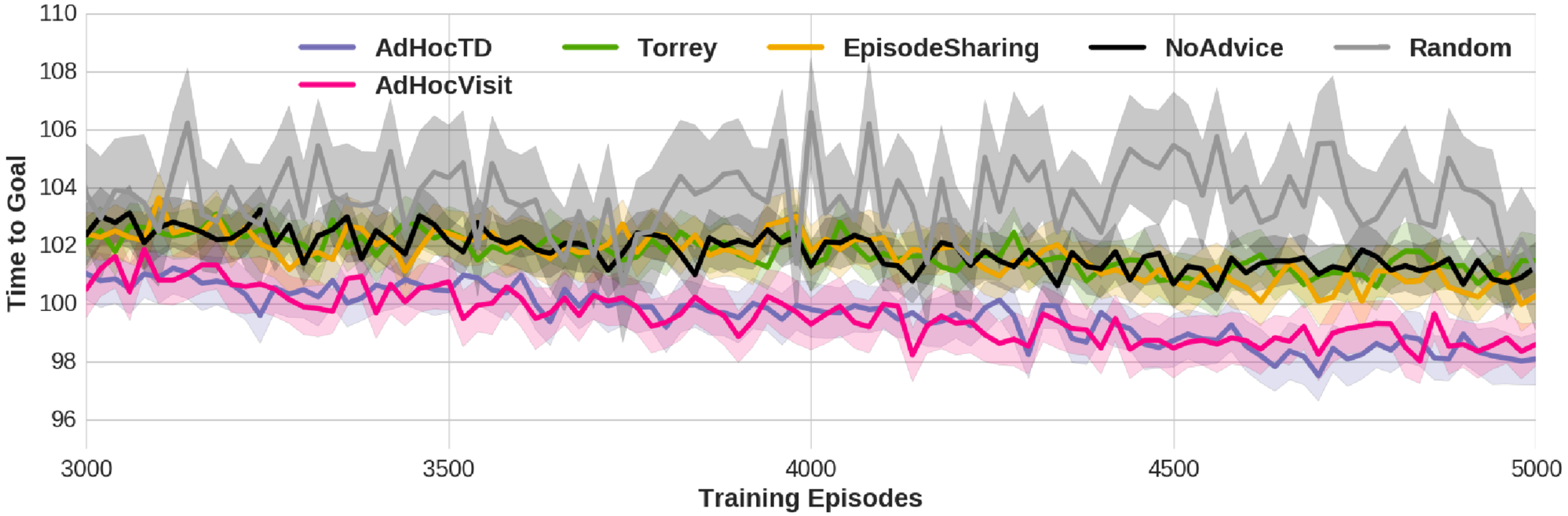
- Goal Percentage (GP)
    - evaluation的时候跑的100个episode里，得分了的episode在100个episode里的占比
- Time to Goal (TG)
    - 平均要多少个step来得分

实验

实验结果 TG

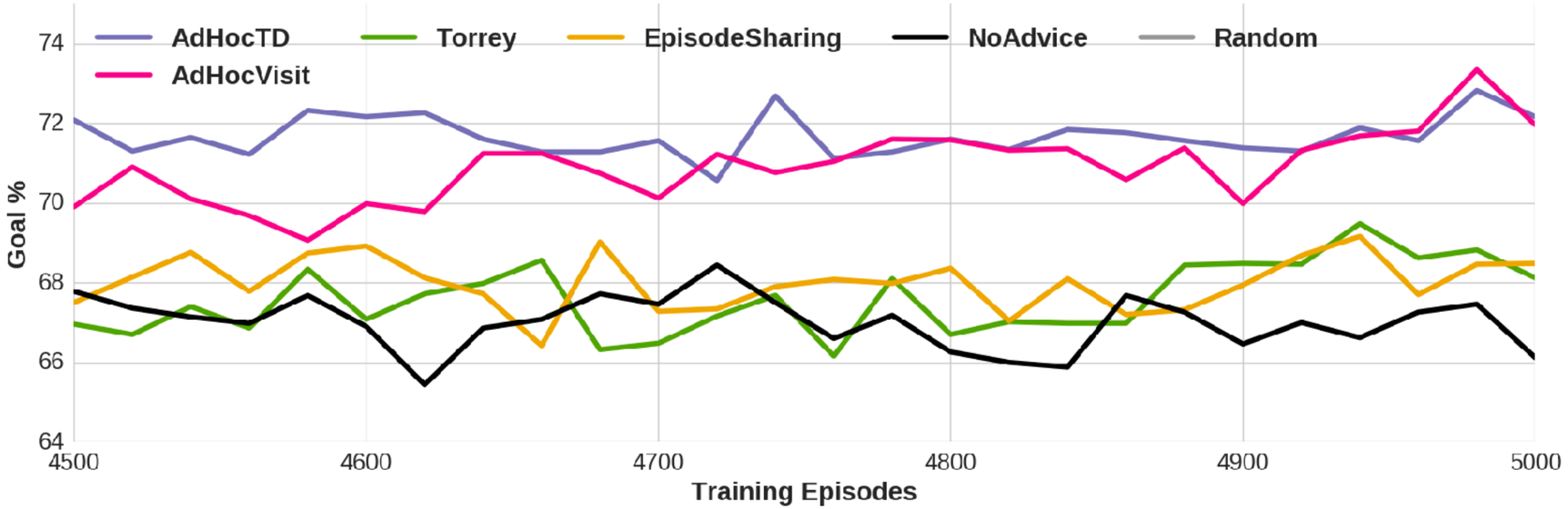
- 一开始训练的时候很快就进球了
- 因为进球的基本都是因为长传然后进了的

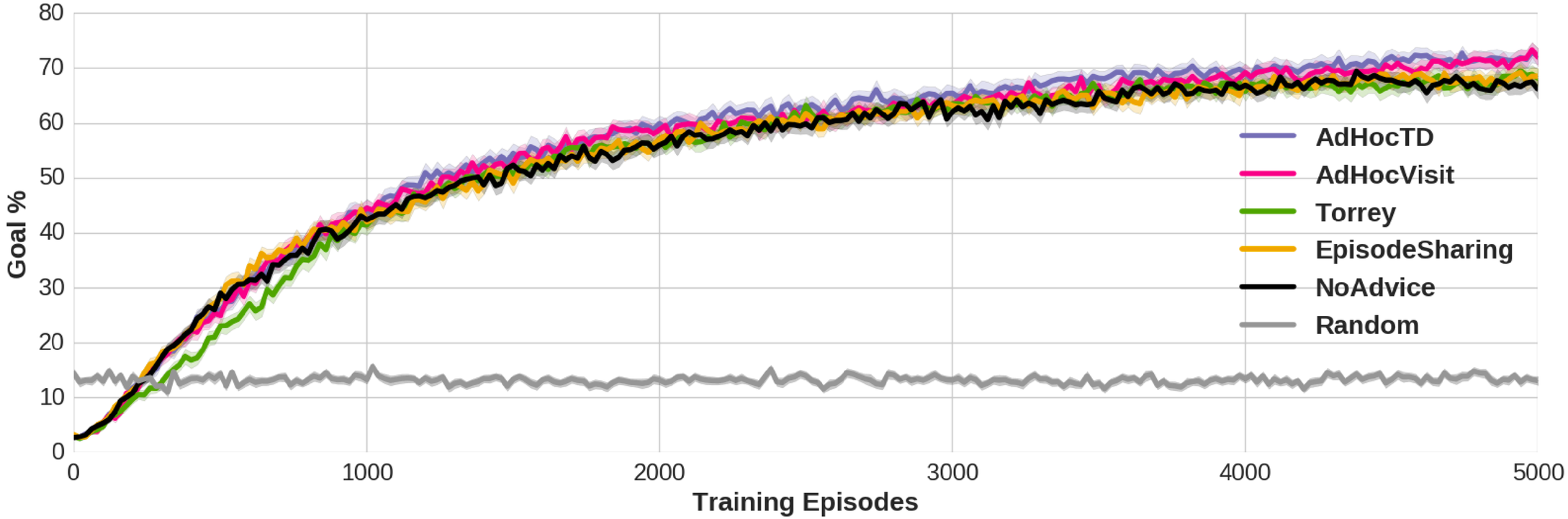




实验

实验结果 GP





- 但是一开始进球率低

实验

实验结果 用掉的budgets